XML, Servicios Web y Web Semántica

Departamento de Informática
Universidad de Oviedo

Sesión 7
Programación XML

SAX, DOM, Enlaces

Departamento de Informática
Universidad de Oviedo

NOTA: Gran parte de estas trasparencias han sido realizadas por César F. Acebal

Programación XML

Los programas que manipulan XML realizan funciones similares:

- Análisis del documento XML
  - Sintáctico y Semántico
- Navegación por el documento
- Transformación del documento
- Generación de nuevos documentos XML

Como XML deriva de SGML desde el principio se dispuso de herramientas

- Toda la tecnología SGML es válida aquí
- Ahora ya hay herramientas específicas para XML

Programación XML

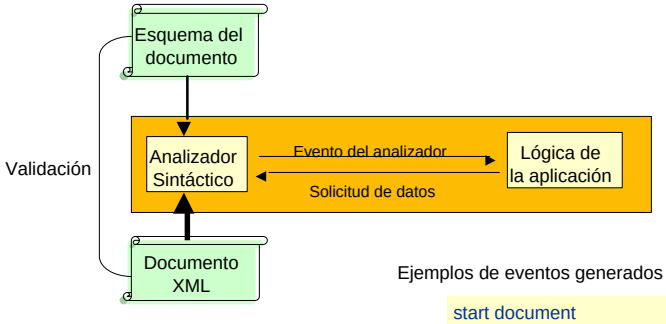
Principales métodos:

- Guiada por eventos: SAX
- Basada en el recorrido del árbol: DOM
- Enlaces (*Bindings*): JAXB

Programación XML
SAX: Modelos de eventos

SAX (Simple API for XML) es un analizador basado en eventos

Se define un modelo de eventos que se generan al analizar el documento XML



Ejemplos de eventos generados

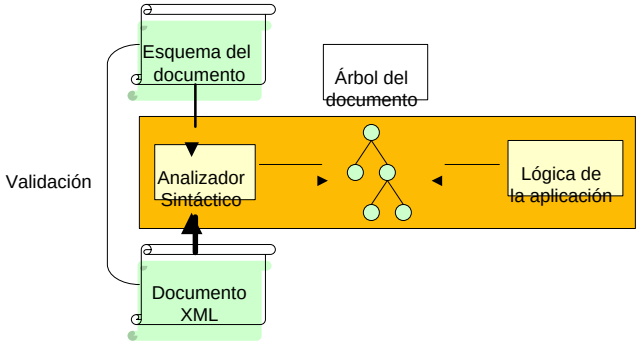
```
start document
start element: pizzas
 start element: pizza
   start element: ingrediente
   end element: ingrediente
   . . .
. . .
```

Procesamiento de XML
DOM: Modelos Basados en Árbol

DOM (Document Object Model)

Se genera un árbol a partir del documento XML

DOM = Interfaz que define cómo recorrer/modificar dicho árbol



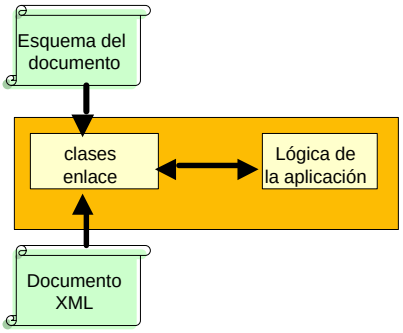
Problemas: Ficheros grandes

Solución: Deferred document (evaluación perezosa)

Procesamiento de XML
JAXB: Enlaces

JAXB: Java API for XML Binding

- A partir del esquema del documento se generan clases de enlace
- Permite trabajar con objetos de la aplicación
- Generación automática de documentos válidos



Modelo basado en eventos: SAX

SAX

SAX: API simple para XML

Creada por desarrolladores XML a partir de los comentarios en la lista xml-dev en 1998

Funcionamiento:

Durante el reconocimiento del documento...

...cada vez que se identifica una estructura (elemento) se mira si hay un procedimiento que manipula ese elemento

- se llama al procedimiento
- cuando termina, continua con el reconocimiento

Los procedimientos permiten procesar el documento guiado por eventos

Los eventos son las ocurrencias de los elementos a los que hacen referencia.

Los datos son los elementos asociados con el elemento identificado

SAX

Manejadores de contenido

Son los procedimientos que se invocan cuando se identifica un elemento.

El código solo se ejecuta cuando el analizador identifica el elemento correspondiente.

Esto permite no tener que cargar todo el documento en memoria.

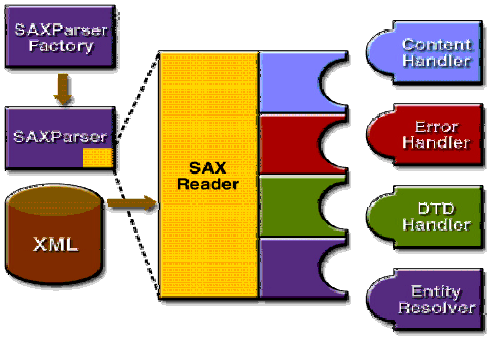
Es la tecnología más eficiente de manipulación de documentos XML

Sobre todo para documentos muy extensos.

SAX
Manejadores de contenido

SAX define cuatro manejadores de contenido

Todos los demás los tiene que construir el programador de la aplicación



SAX
Manejadores estándar

ContentHandler
- permite manejar eventos relacionados con los datos de un documento XML

ErrorHandler
- recibe información sobre errores en los datos del documento XML

DTDHandler
- permite manejar eventos relacionados con el análisis de un DTD. No atiende a eventos relacionados con la validación. Poco usado

EntityResolver
- analiza entidades externas que se especifican en el documento XML que se está analizando.

SAX
Manejadores de Contenido

Localizador de documento
- Se llama siempre que ocurre un evento.
- Permite conocer qué parte del documento XML se está analizando
  - (línea y columna)

Comienzo del documento:
- startDocument()
- Indica el comienzo del análisis.
- Es el primero de todos los métodos que se llama.

Fin del documento:
- endDocument()
- Indica el fin del análisis.
- Es el último de todos los métodos que se llama.

SAX
Manejadores de Contenido

Instrucciones de proceso
- Son directivas para la aplicación que está trabajando con el documento XML y no se consideran elementos XML
  - Cocoon las utiliza para indicar como presentar los datos una vez analizados.
- Informa del destino de la instrucción de proceso (target) y los datos (data)

Espacio de nombres
- Indica el comienzo y el fin de un espacio de nombres en el documento XML
- Informa del prefijo del espacio de nombres y de la URI asociada a ese prefijo.

```
<alias:Elemento1 xmlns:alias="http://mio.com">
        <alias:Elem2>HOLA</alias:Elem2>
</alias:Elemento1>
```

SAX
Manejadores de contenido

- Elemento
  - Indica el inicio y el fin de un elemento XML
  - Informa del nombre del elemento y de los atributos que posee.
    - Permite recorrer los atributos mediante un bucle.
    - Los atributos pueden no estar ordenados.
- Datos del elemento
  - Los elementos anidados dentro de otros elementos generan llamadas recursivas hasta encontrar datos de tipo texto.
  - Informa de los datos mediante una matriz que contiene la información extraída del documento XML
  - No está claramente especificado si los datos se devuelven todos de una vez o mediante llamadas sucesivas.

SAX
Manejadores de errores

- Sirve para tratar los errores que se puedan producir durante el análisis.
- Cada llamada recibe la información a través de la excepción SAXParseException
  - Contiene el número de línea.
  - La URI del documento XML que se está tratando.
  - Detalles de la excepción.
    - Mensaje explicativo.
    - Traza de la pila.
- 3 tipos: Avisos (*warning*), errores y errores fatales

SAX
Ejemplo:SimpleSAX

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;

public class SimpleSAX extends DefaultHandler {

  public void startElement(String ns, String local, String qName, Attributes atts) {
     System.out.println("startElement: " + local);
  }

 public static void main(String[] args) throws SAXException {

   SimpleSAX s = new SimpleSAX();
   SAXParser p = new SAXParser();
   p.setContentHandler(s);

   try {
    p.parse(args[0]);
   } catch (Exception e) {e.getMessage(); }}
}
```

SAX
ContentHandler

Principales métodos que define esta interfaz :

- setLocator
- startDocument
- endDocument
- startElement
- endElement
- characters
- ...etcétera

SAX
Locator

Proporciona una serie de métodos como **getLineNumber** y **getColumnNumber** que permiten saber la ubicación actual en el fichero XML donde tuvo lugar el último evento

Sólo tiene validez dentro de un **ContentHandler**

Se suele asignar a un atributo de la clase

```
private Locator locator;

public void setDocumentLocator(Locator locator)
{
   this.locator = locator;
}
```

SAX
Principio y final de un documento

Se corresponden con los eventos **startDocument** y **endDocument**, respectivamente

Son siempre el primer y el último método llamados, no sólo dentro del **ContentHandler**, sino en el proceso de análisis completo

Incluso en la presencia de otros manejadores, como DTDHandler

SAX
Una nota sobre las excepciones

Muchos de los métodos de respuesta a eventos SAX, como los dos anteriores, declaran una excepción **SAXException**

Es el único tipo de excepción que puede lanzar un analizador SAX

- Cualquier otra excepción será encapsulada en ésta
  - Por ejemplo, si se procesa un documento a través de la red y se produce una excepción de entrada/salida (IOException)
- A través del método `getException` se puede acceder a la excepción subyacente

SAX
Elementos

Hay tres eventos SAX que nos permiten acceder al contenido en sí del documento analizado:

- El principio y final de los elementos y el método `characters`

startElement proporciona información sobre un elemento XML:

- Nombre
- Espacio de nombres, si lo hubiera
- Atributos: En un objeto `org.xml.sax.Attributes`
  - Similar a una lista o vector

endElement es muy similar al anterior, pero para el final del elemento

SAX
Contenido de un elemento

Una vez que hemos visto cómo saber el principio y fin de un elemento y los atributos de éste, es necesario poder acceder al contenido del mismo

Puede estar formado por elementos hijos, información textual, o una combinación de ambos

A dicha información textual se accede mediante el método **characters**, en forma de un array de caracteres

```
void characters(char[] ch, int start, int length)
```

SAX
Sobre el método characters

SAX no define cómo se devolverán grandes porciones de datos textuales a través de este método

Puede ser en una única llamada, pero también podrían partirse en varias invocaciones al método characters

Y hay también que tener en cuenta el carácter secuencial del procesamiento mediante SAX

¿Qué eventos tendrán lugar para el siguiente fragmento de XML?:

```
<padre>Esto es<hijo>¡hola!</hijo>un ejemplo</padre>
```

Ejemplo: videoclub.xml

Sea el siguiente documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<videoclub>
  <pelicula titulo="2001: una odisea del espacio" año="1968" duracion="143">
    <guion>
      <guionista>Stanley Kubrick</guionista>
      <guionista>Arthur C. Clarke</guionista>
    </guion>
    <direccion>
      <director>Stanley Kubrick</director>
    </direccion>
    ...
    <sinopsis>"2001: Una Odisea del Espacio" es una cuenta atras hacia el futuro,
un mapa del destino humano, una conquista del infinito, un apasionante drama
sobre el hombre que lucha contra la maquina.</sinopsis>
  </pelicula>
  <pelicula titulo="Alien resurreccion" año="1998" duracion="105">
    ...
  </pelicula>
  ...
</videoclub>
```

Ejemplo: videoclub.xml

Hacer un programa que muestre la siguiente información acerca de cada película:

Título, año de rodaje y duración

Como esa información está disponible en los atributos del elemento **<pelicula>**, bastará con redefinir el método **startElement**

Segundo ejemplo

Ahora, además de los datos anteriores, mostrar la sinopsis de la película

¿Cómo?

- Necesitamos obtener el contenido del elemento <sinopsis>
- Para ello necesitamos redefinir el método characters
  - Pero sólo nos vale cuando se esté analizando ese elemento concreto

Sugerencia:

- Atributo que nos diga cuándo estamos dentro del elemento <sinopsis>
  - Redefinir los métodos startElement y endElement
- Atributo en el que iremos almacenando los caracteres
  - Recordemos que puede haber varias llamadas a characters para un elemento dado

SAX ErrorHandler

Define tres métodos para el tratamiento de los distintos tipos de errores:

- void warning(SAXParserException)
- void error(SAXParserException)
- void fatalError(SAXParserException)

Todos ellos reciben información sobre el error ocurrido a través de un objeto **SAXParserException**

- Contiene el número de línea, el URI del documento analizado, un mensaje con la causa del error, etcétera
- Cada uno puede lanzar a su vez una excepción SAXException
  - (Si se produjese una excepción durante el tratamiento del error)

Tipos de error

- La mayoría de los avisos y los errores normales tienen que ver con la validación de los documentos
- Los errores fatales suelen ser relativos a la mala formación del documento
  - También con la versión de XML del documento:
    - `<?xml version="1.2"?>`
  - Ejercicio: crear un manejador `ErrorHandler` y probar algunos errores fatales (`SAXParserErrorDemo.java`)
    - Si el documento es inválido, ¿se producen errores?
      - No, porque por omisión el analizador no valida (hay que indicárselo)

SAX
Configuración del analizador

SAX 2.0 define un mecanismo estándar para configurar ciertas propiedades de los analizadores de forma que sea fácil añadir otras nuevas a medida que sean aceptadas por el W3C sin tener que usar extensiones o métodos propietarios de cada implementación

Método	Devuelve	Parámetros
setProperty	**void**	**String propertyID, Object value**
setFeature	**void**	**String featureID, boolean state**
getProperty	**Object**	**String propertyID**
getFeature	**boolean**	**String featureID**

SAX
Propiedades y funcionalidades

Una propiedad (`property`) necesita de algún objeto para que pueda ser utilizada

Una funcionalidad (`feature`), por el contrario, no es más que un interruptor empleado por el analizador para determinar si está o no activado un tipo determinado de procesamiento

Por ejemplo, si debe validar el documento

SAX
Activar la validación

En SAX 2.0, basta con la siguiente línea:

```
parser.setFeature("http://xml.org/sax/features/validation", true)
```

En Xerces, esto activa la validación frente a un DTD

Si quisiéramos validar frente a un esquema, habría que activar también la siguiente *feature*:

```
"http://apache.org/xml/features/validation/schema"
```

DOM

Modelo de Objetos de Documento

DOM
Orígenes

DOM: Modelo de Objetos de Documento

Origen en el Consorcio World Wide Web (W3C)

Es una interfaz independiente de la plataforma y el lenguaje de programación que permite acceder y manipular dinámicamente el contenido, el estilo y la estructura de un documento.

La norma se define en niveles en lugar de versiones:

- Nivel 1: describe la funcionalidad y el modo de navegar por un modelo general
- Nivel 2: se centra en tipos de documentos específicos (XML, HTML, CSS)
- Nivel 3: mas facilidades en tipos de documentos específicos (validación para XML)

DOM
Características

DOM ofrece una interfaz genérica descrita en IDL

- Independiente del lenguaje
- Independiente del fabricante

Es responsabilidad del desarrollador producir la implementación para el lenguaje deseado

- Java, C++, Perl, Python,....

El modelo está basado en objetos

DOM
Partes de DOM

El núcleo fundamental

- Proporciona la interfaz básica que puede representar cualquier documento XML

HTML

- Extensión para manipular documentos HTML

DOM
Elementos de DOM

La base de DOM es un árbol de nodos.

No se especifica como tiene que ser la implementación de ese árbol de nodos.

Solo tiene que cumplir una interfaz

Tipos de nodos

- Document
- DocumentFragment
- Element
- Attr
- ProcessingInstruction
- Text
- Comment
- CDATASection
- EntityReference
- Entity
- Notation
- DocumentType

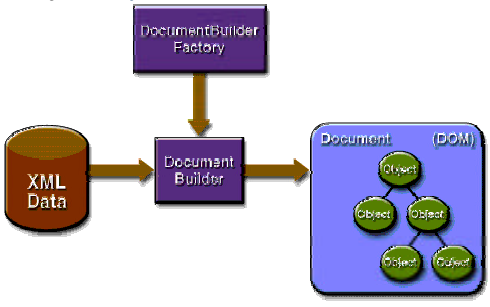
DOM
Estructura jerárquica

En DOM, un documento XML se representa como un árbol

Un analizador DOM genera automáticamente dicho árbol a partir de un documento XML

Crea la estructura completa en memoria, de forma que puedan utilizarla las aplicaciones que trabajen con el documento

DOM proporciona los métodos, clases e interfaces necesarios para acceder al árbol y manipular su estructura o modificar sus elementos

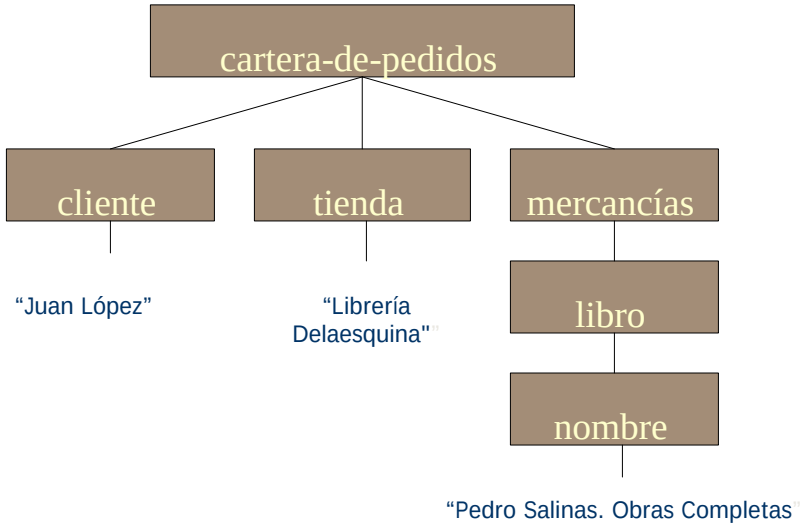


Ejemplo

¿Qué genera DOM para el siguiente documento?

```
<?xml version="1.0"?>
<cartera-de-pedidos>
  <cliente>Juan López</cliente>
  <tienda>Librería Delaesquina</tienda>
    <mercancías>
      <libro>
        <titulo>Pedro Salinas. Obras completas</titulo>
      </libro>
    </mercancías>
  </tienda>
</cartera-de-pedidos>
```

Árbol generado



Creación del analizador

A diferencia de SAX, DOM sólo define las clases e interfaces necesarias para representar y manipular el documento como un árbol

No hay, por tanto, una clase estándar para el analizador

Deberemos crear una instancia de la proporcionada por nuestra implementación concreta (en este caso, Xerces)

```
DOMParser parser = new DOMParser();
```

Definida en org.apache.xerces.parsers

DOM
Ejemplo Simple

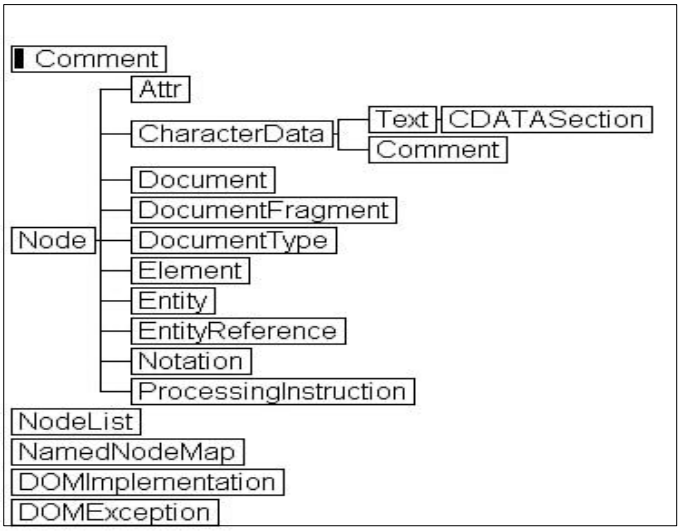
```
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;

public class SimpleDOM {
  public SimpleDOM(String uri) throws Exception
  {
    System.out.println("Analizando el documento: " + uri + "...");
    DOMParser parser = new DOMParser();
    parser.parse(uri);
    Document document = parser.getDocument();
  }

  public static void main(String[] args) throws Exception
  {
    if (args.length != 1) {
      System.out.println("Modo de empleo: SimpleDOM fichXML ");
      System.exit(0);
    }
    new SimpleDOM(args[0]);
  }
}
```

DOM
Resultado del análisis: árbol DOM

¿Cómo accedemos a la estructura resultante del análisis DOM?

- El método **parse** no devuelve nada
- Es necesario llamar a **getDocument**
  - Devuelve un objeto **org.w3c.dom.Document**

DOM
Jerarquía de clases



DOM
Node

Es la interfaz base de todo el modelo de objetos de documento

Representa un nodo cualquiera del árbol DOM

Provee una serie de métodos para acceder al nombre del nodo actual, sus atributos, sus hijos (si los tuviera), etc.

También indica el tipo de nodo concreto de que se trata (si es un elemento, un atributo, una entidad…)

Lo hace a través del método **getNodeType** y de unas *constantes* públicas definidas en esta clase

DOM
Conocer el tipo de nodo

El pseudocódigo habitual en cualquier procesamiento DOM es algo similar a esto:

```
tratarNodo(Node nodo)
{
   switch (nodo.getNodeType())
   {
      case Node.DOCUMENT_NODE: ...
      case Node.ELEMENT_NODE: ...
      case Node.ATTRIBUTE_NODE: ...
      ...
   }
}
```

DOM Document

Representa al documento completo

Es un caso particular de nodo, ya que contiene tanto el elemento raíz como el DTD del documento y más información especial relativa al documento completo y que no está en la jerarquía del mismo

Mediante **getDocumentElement** se accede al elemento raíz

```
case Node.DOCUMENT_NODE:
    System.out.println("<?xml version=\"1.0\"?>");
    Document doc = (Document) nodo;
    hacerAlgo(doc.getDocumentElement());
    break;
```

Suele ser habitual este tipo de *downcast* al trabajar con DOM y Node

DOM Element

Una de las tareas más comunes será trabajar con elementos DOM

- Tratar su nombre y atributos, o su valor
  - getNodeName()
  - getNodeValue()
  - getNodeAttributes()
    - Devuelve un instancia de NamedNodeMap
      - Representa una colección de nodos que son únicos con respecto al nombre
- A continuación hacer lo mismo para todos sus hijos
  - getChildNodes()
    - Devuelve un instancia de NodeList

DOM
Ejemplo: videoclub.xml

Hacer el mismo ejercicio que habíamos visto con SAX utilizando DOM:

- Primero, mostrar sólo el título, año y duración de cada película
  - Hacerlo primero con un enfoque similar al anterior
  - Luego, probar a simplificarlo haciendo uso del método `getElementsByTagName`, de `Node`
- Luego, mostrar también la sinopsis
  - Recuérdese que el contenido de un elemento está disponible como un nodo hijo del mismo (de tipo `Text`)

DOM
Creación del documento

La API de DOM define la interfaz **DOMImplementation** que permite crear un objeto **Document**

- Cada analizador concreto deberá implementar dicha interfaz
  - En el caso de Xerces lo hace la clase `org.apache.xerces.dom.DOMImplementationImpl`
- Éste es el método que permite crear el documento:

```
public Document createDocument(String namespaceURI,
                               String qualifiedName,
                               DocumentType doctype)
throws DOMException
```

DOM
Creación del documento

```
DOMImplementation domImpl = new DOMImplementationImpl();
Document doc =
      domImpl.createDocument(null, "empleados", null);
```

Las dos líneas anteriores crean, en Xerces, un objeto **Document** "a mano"

Ahora ya podemos acceder a los métodos de fabricación de **Document** para crear elementos, atributos, etc.

DOM
Ejemplo de creación de un árbol

Crear un árbol DOM que represente el siguiente documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<profesores>
  <persona>
    <nombre>César F. Acebal</nombre>
    <correo-e>acebal@uniovi.es</correo-e>
    <url>www.cfacebal.com</url>
  </persona>
  <persona>
    <nombre>José E. Labra Gayo</nombre>
    <correo-e>labra@lsi.uniovi.es</correo-e>
    <url>www.di.uniovi.es/~labra</url>
  </persona>
</profesores>
```

Otras API

Para concluir con el procesamiento de documentos XML, veremos otras dos interfaces de programación de aplicaciones (API) específicas del lenguaje Java y que se sitúan, por tanto, a un mayor nivel de abstracción que SAX y DOM: JAXP y JDOM.

Java API for XML Processing (JAXP)

Desarrollada por Sun, esta sencilla API proporciona una capa de abstracción sobre SAX o DOM para hacer independiente a la aplicación de la implementación concreta del analizador utilizado

http://java.sun.com/xml/jaxp/index.jsp

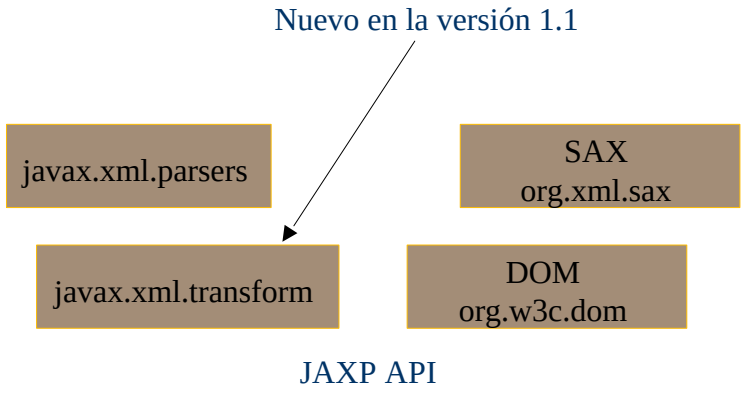
Soporta:

- Análisis de documentos: SAX 2.0 y DOM Level 2
- Transformación mediante XSLT

Se incluye en la *Java Standard Edition (JSE)*

- `javax.xml.*`

JAXP
Panorámica de JAXP



JAXP
Obtención de un analizador SAX con JAXP

```
javax.xml.parsers.SAXParserFactory parserFactory =
        SAXParserFactory.newInstance();
// Configuración de la fábrica
parserFactory.setValidating(true);
...
javax.xml.parsers.SAXParser parser =
        parserFactory.newSAXParser();
```

SAXParser es una clase envoltorio (***wrapper***), abstracta, de **org.sax.XMLReader**

JAXP

¿Qué implementación se selecciona?

- Se configura mediante una propiedad del sistema
  - javax.xml.parsers.SAXParserFactory
- Lo único que hace es aislar al código de cualquier dependencia con un analizador SAX concreto (Xerces o cualquier otro)

JAXP
Obtención de un analizador DOM

El uso de JAXP para obtener una implementación concreta de un analizador DOM sigue los mismos principios que acabamos de ver para SAX

- También en el paquete javax.xml.parsers están las siguientes dos clases:
  - DocumentBuilder y DocumentBuilderFactory
- Al igual que con SAX, la implementación concreta se selecciona a través de una propiedad del sistema:
  - javax.xml.parsers.DocumentBuilderFactory

JAXP
Obtención de un analizador DOM

```
DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
// Configuración de la fábrica
factory.setValidating(true);
...
DocumentBuilder parser = factory.newDocumentBuilder();
Document doc = parser.parser(uri);
```

JDOM

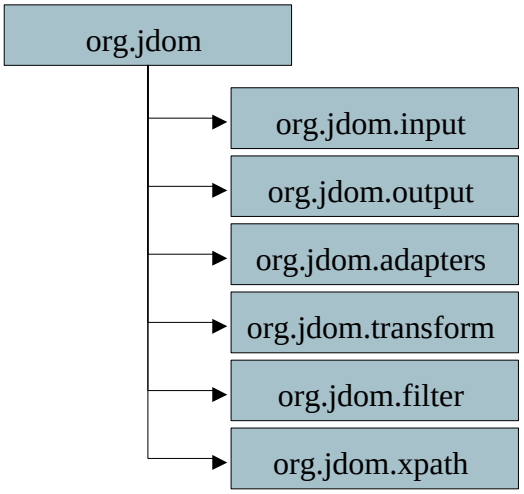
Trata de resolver las deficiencias de SAX, DOM y JAXP

Es una API específica del lenguaje Java, y se sitúa a un nivel de abstracción mayor que las otras

- www.jdom.org
- Es más cómodo crear un documento XML a mano, desde Java, empleando JDOM que DOM

Es similar a DOM en el sentido de que también genera un árbol, pero no está construido sobre él

JDOM
Estructura de paquetes



JDOM
Element. *Algunos ejemplos*

Algunos ejemplos de cómo se accede al contenido de un elemento:

```
// Obtener el elemento raíz
Element root = doc.getRootElement();
// Obtener una lista con todos sus hijos
List allChildren = root.getChildren();
// Obtener los hijos de un nombre dado
List namedChildren = root.getChildren("nombre");
// Sólo el primer elemento que concuerde con un nombre
List child = root.getChild("nombre");
```

Además, ¡la lista es una List de Java!

JDOM
Contenido de un elemento

```
<desc>
  Una demo
</desc>
```

El texto está disponible directamente

En DOM es necesario acceder al elemento hijo de tipo `Text`, podía haber más de uno…

```
String desc = element.getText();
```

Devuelve `"\n Una demo \n"`

Hay un método más práctico:

```
String desc = element.getTextTrim();
```

Devuelve `"Una demo"`

JAXB
Modelo basado en enlaces

Facilita la correspondencia entre documentos XML y clases Java, y viceversa

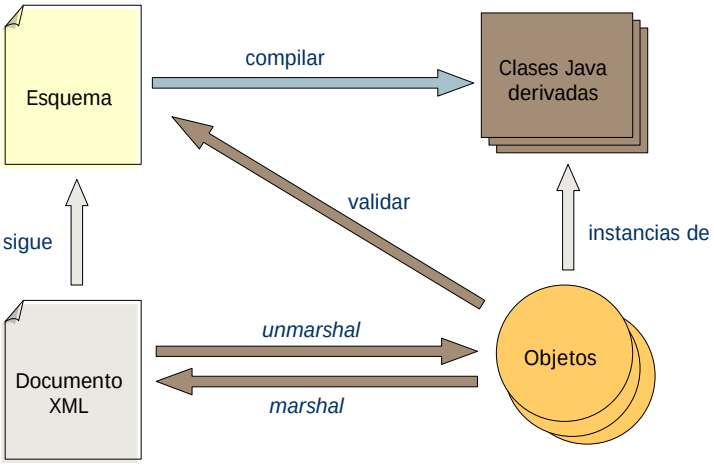
Incluye un compilador y un entorno en tiempo de ejecución

- El compilador traslada un esquema de XML a clases Java automáticamente
- El entorno en tiempo de ejecución es capaz de validar dichas clases frente a un esquema y generar, por tanto, documentos XML válidos

Página inicial de JAXB:

- http://java.sun.com/xml/jaxb/
  - Hay una implementación de JAXB incluida en el Java Web Services Developer Pack

JAXB
Funcionamiento



JAXB
Ejemplo

```
<xs:schema ... >
<xs:element name="alumnos">
  <xs:complexType>
   <xs:sequence>
     <xs:element name="alumno" minOccurs="1" maxOccurs="200"
       type="TipoAlumno"/>
    </xs:sequence>
   </xs:complexType>
</xs:element>

 <xs:complexType name="TipoAlumno">
    <xs:sequence>
       <xs:element name="nombre" type="xs:string"/>
       <xs:element name="apellidos" type="xs:string"/>
       <xs:element name="nota"
                    minOccurs="0" maxOccurs="1" type="xs:float"/>
       <xs:element name="email"
                    minOccurs="0" maxOccurs="1" type="xs:anyURI"/>
    </xs:sequence>
    <xs:attribute name="dni" type="xs:string"/>
  </xs:complexType>
</xs:schema>
```

JAXB
Ejemplo

Generar clases a partir del esquema (-p = nombre paquete)

```
> xjc alumnos.xsd -p alumnos
parsing a schema...
compiling a schema...
alumnos\impl\runtime\SAXUnmarshallerHandler.java
alumnos\impl\runtime\UnmarshallerImpl.java
. . .
```

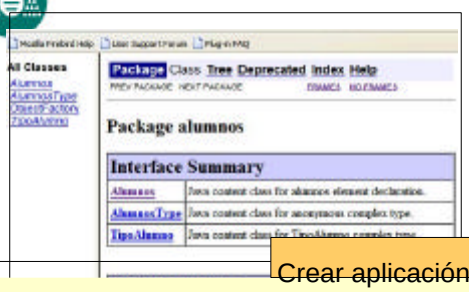
Compilar clases

```
> javac alumnos/*.java
```

Opcionalmente, puede generarse documentación Javadoc

```
> javadoc -package alumnos -sourcepath . -d docs/api -windowtitle "alumnos"
```

JAXB
Ejemplo



Crear aplicación

```
public class Main {
 public static void main( String[] args ) {
       . . .
   try {
     JAXBContext jc = JAXBContext.newInstance( "alumnos" );
     Unmarshaller u = jc.createUnmarshaller();

     Alumnos als = (Alumnos) u.unmarshal(new FileInputStream(uri) );
    TipoAlumno a = (TipoAlumno) als.getAlumno().get(0);
    for (Iterator it=als.getAlumno().iterator(); it.hasNext(); ) {
       TipoAlumno a = (TipoAlumno) it.next();
       System.out.println("Alumno dni = " + a.getDni() + ", nota = " +a.getNota());
    }
 . . .
```

Fin